Gu, Feng and Greensmith, Julie and Aickelin, Uwe (2013) Theoretical formulation and analysis of the deterministic dendritic cell algorithm. Biosystems, 111 (2). pp. 127-135. ISSN 0303-2647

Theoretical Formulation and Analysis of the Deterministic Dendritic Cell Algorithm

Feng Gu[a], Julie Greensmith and Uwe Aickelin[b]

[a]*School of Computing, University of Leeds, LS2 9JT, UK*
[b]*School of Computer Science, University of Nottingham, NG8 1BB, UK*

Abstract

As one of the emerging algorithms in the field of Artificial Immune Systems (AIS), the Dendritic Cell Algorithm (DCA) has been successfully applied to a number of challenging real-world problems. However, one criticism is the lack of a formal definition, which could result in ambiguity for understanding the algorithm. Moreover, previous investigations have mainly focused on its empirical aspects. Therefore, it is necessary to provide a formal definition of the algorithm, as well as to perform runtime analyses to reveal its theoretical aspects. In this paper, we define the deterministic version of the DCA, named the dDCA, using set theory and mathematical functions. Runtime analyses of the standard algorithm and the one with additional segmentation are performed. Our analysis suggests that the standard dDCA has a runtime complexity of $\mathcal{O}(n^2)$ for the worst-case scenario, where n is the number of input data instances. The introduction of segmentation changes the algorithm's worst case runtime complexity to $\mathcal{O}(\max(nN, nz))$, for DC population size N with size of each segment z. Finally, two runtime variables of the algorithm are formulated based on the input data, to understand its runtime behaviour as guidelines for further development.

Key words: Artificial Immune Systems, Dendritic Cell Algorithm, Runtime Analysis, Formulation and Formalisation

1. Introduction

Artificial Immune Systems (AIS) [7, 18] are computer systems inspired by both theoretical immunology and observed immune functions, principles and models, which are applied to real-world problems. The human immune

system from which AIS draw inspiration, is evolved to protect the host from a wealth of invading micro-organisms. AIS are developed to provide similar defensive properties within a computing context. Initially, AIS were based on simple models of the human immune system. As noted by Stibor *et al.* [29], 'first generation' immune algorithms, such as negative selection and clonal selection, do not produce the same high-quality performance as the human immune system. These algorithms, negative selection in particular, are prone to problems with scalability and the generation of excessive false alarms, when used to solve problems such as network-based intrusion detection. Recent AIS use more rigorous and up-to-date immunology and are developed in collaboration with modellers and immunologists. The resulting algorithms are believed to encapsulate the desirable properties of immune systems, including robustness, error tolerance, and self-organisation [7].

One such 'second generation' immune algorithms is the Dendritic Cell Algorithm (DCA) [10]. The algorithm is inspired by functions of the dendritic cells (DCs) of the innate immune system, while incorporating principles of a key novel theory in immunology, named the *danger theory* [21]. An abstract model of natural DC behaviour is used as the foundation of the developed algorithm. The DCA has been successfully applied to numerous security-related problems, including port scan detection [10], botnet detection [1] and as a classifier for robot security [25]. These applications refer to the area of anomaly detection, which is essentially one particular type of binary classification with an 'anomalous' class and a 'normal' class. According to results of these applications, the DCA has shown not only good performance in terms of detection rate, but also the ability to reduce the rate of false alarms in comparison to other systems, such as Self Organising Maps (SOM) [13].

However, there are also issues concerning the DCA. One criticism is the lack of a formal definition, which could result in ambiguity for understanding the algorithm and thus lead to incorrect applications and implementations. It is pointed out in [28] that the DCA shares similarities to linear classifiers since it employs a linear discriminant function for signal transformation. However, the DCA is not simply a collection of linear classifiers, as it performs classification based on the temporal correlation of a multi-agent DC population, rather than linear signal transformation. Signal transformation is used to identify if any anomalies occurred in the past. Whether the identified anomalies can be correctly correlated with potential causes is determined by the effectiveness of the temporal correlation performed at the population level. As a first step, a formal definition should be provided for presenting

the algorithm in a clear and accessible manner.

Previous investigations have mainly focused on its empirical aspects, evidenced by experimental results on a range of problem domains. Except for the geometry analysis of Stibor *et al.* [28] that was later extended in Oates's thesis [24], theoretical analysis of the DCA has barely been performed, and most theoretical aspects of the algorithm have not yet been revealed. Other immune inspired algorithms, such as negative and clonal selection algorithms, were theoretically presented in [30]. Elberfeld and Textor [9] theoretically analysed string-based negative selection algorithms, to show the possibility of reducing the worst-case runtime complexity from exponential to polynomial, through compressing detectors. More recently, the work of Zarges [31, 32] theoretically analysed one of the vital components of the clonal selection based algorithms, namely inversely proportional mutation rates. Jansen and Zarges [19] performed a theoretical analysis of immune inspired somatic contiguous hypermutations for function optimisation. As a result, it is important to conduct a similar theoretical analysis of the DCA, to determine its runtime complexity and numerous other algorithmic properties, in line with other AIS.

In this paper, we extend the work presented in [15], which involved formal specifications of a single-cell model at the behavioural level using interval temporal logic [23]. Note the algorithm demonstrated in this work is the deterministic DCA (dDCA) [12], created by removing stochastic components for the ease of analysis. Any statements of the DCA made subsequently are referred to the dDCA. The aim is to provide a clear and accessible definition of the DCA, as well as an initial theoretical analysis on the algorithm's runtime complexity and other algorithmic properties. As potential readers may not have a deep understanding of complicated formal methods such as the B-method [20], we use set theory and mathematical functions to specify the algorithm. From the formal definitions, theoretical analyses on the runtime complexity are performed, for the standard algorithm and an extended system with segmentation. Moreover, the formulations of two important runtime variables are included to present the algorithm's runtime behaviour, and to provide guidelines for future development. The paper is organised as follows, an overview of the DCA is given in Section 2, the formal definition is presented in Section 3, runtime analyses are shown in Section 4, formulation of two runtime variables is described in Section 5, and finally conclusions and future work are presented in Section 6.

2. The Dendritic Cell Algorithm

2.1. Biological Background

The DCA is inspired by functions of the dendritic cells (DCs) of the innate immune system, which forms part of the body's first line of defence against invaders. DCs exhibit the ability to combine a multitude of molecular information and to interpret this information for the T-cells of the adaptive immune system. This could lead to the induction of various immune responses against perceived pathogenic threats. Therefore, DCs are often seen as detectors responsible for policing different tissues, as well as inductive mediators for a variety of immune responses.

In general, two types of molecular information are processed by DCs, namely 'signal' and 'antigen'. Signals are collected by DCs from their local environment and consist of indicators of the health status of the monitored tissue. Throughout its lifespan, an individual DC will exist in one of three states, namely 'immature', 'semi-mature' and fully 'mature', as shown in Figure 1. In the initial immature state, DCs are exposed to a combination of signals, and perform phagocytosis to ingest substances from their surroundings. Based on the concentration of presented signals, DCs differentiate into either a 'fully mature' form to activate the adaptive immune system, or a 'semi-mature' form to suppress it. If a DC is exposed to a combination of signals generated from a healthy or steady state tissue environment, such as no occurrence of tissue damage, it more likely becomes a semi-mature DC. Conversely, if a DC is presented with a combination of signals generated from a damaged tissue environment, such as the presence of unregulated cell death, it more likely differentiates into a fully mature DC. Natural DCs bind to and process many cytokine signals. In an abstract model of DC behaviour developed by Greensmith [10], the following categories are defined.

- **PAMP**: Pathogenetic Associated Molecular Patterns, molecular signatures of pathogens which are recognised by Toll-Like Receptors (TLRs) on the surface of DCs, and they are highly influential to the transition from immature state to fully mature state;

- **Danger**: released by damaged tissue cells subject to necrosis (unregulated cell death), they have a lower effect than PAMPs on the maturation towards fully mature state;

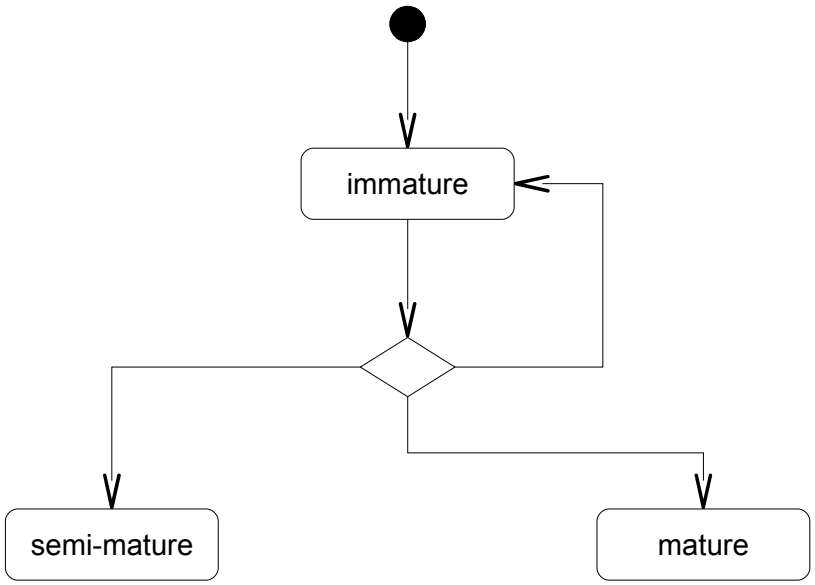

Figure 1: A state-chart describing the three states of an individual DC.

- **Safe** signals are derived from the cells that encounter apoptosis (programmed cells death), TNF-α (Tumour Necrosis Factor) is one candidate of safe signals, they contribute to the maturation from immature state to semi-mature state;

During the immature state, DCs also collect debris in the tissues which are subsequently combined with the environmental signals. Some of the 'suspicious' debris collected are known as antigens, and they are proteins originating from potential invading entities. DCs combine the 'suspect' antigens with evidence in the form of signals to correctly instruct the adaptive immune system to respond, or become tolerant to the presented antigens. For more detailed information regarding the underlying biological mechanisms, please refer to [10, 21].

2.2. Algorithmic Details

The DCA was designed and developed based on an abstract DC model created by Greensmith [10]. It incorporates the functionality of DCs including data fusion, state differentiation and causal correlation. As per the

natural system, there are two types of input data, namely 'antigen' and 'signal'. It is generally assumed that certain causal relationship exists between the two data streams. Antigens are categorical values that can be various states of a problem domain or the entities of interest associated with a monitored system. Signals are represented as vectors of real-valued numbers, and they are measures of a monitored system's status within certain time periods. In real-world applications, antigens represent what is to be classified within a given problem domain. For instance, they can be process IDs in computer security problems [1, 11], a small range of positions and orientations of robots [25], the proximity sensors of online robotic systems [22], or the time stamps of records collected in biometric data [17]. Signals represent system context of a host or a measure of network traffic [1, 11], the readings of various sensors in robotic systems [25, 22], or the biometric data captured from a monitored automobile driver [17]. Signals are normally pre-categorised as 'PAMP','Danger' or 'Safe'. The semantics of these signal categories is listed as follows:

- **PAMP**: increases in value as the observation of anomalous behaviour, it is a confidence indicator of anomaly, which usually is presented as signatures of the events that can definitely cause damage to the system;
- **Danger**: reflects to potential anomalies, as the values increases, the confidence of the abnormal status of the monitored system increases accordingly;
- **Safe**: increases in value in conjunction with observed normal behaviour, this is a confidence indicator of normal, predictable or steady-state system behaviour.

Increases in the value of safe signal suppress the effect of the PAMP and Danger signals within the algorithm, as per what is observed in the natural system. This immunological property is incorporated within the DCA in the form of predefined weights for each signal category, for the transformation from input signals to output signals, which are 'CSM' and 'K' signals. The CSM signal reflects the amount of information a DC has processed, i.e. when to make decisions, while the K signal is a measure indicating the polarisation towards anomaly or normality, i.e. how to make decisions. The output signals are used to evaluate the status of the system monitored by the analysis component of the algorithm. Such a signal transformation process is displayed in Figure 2.

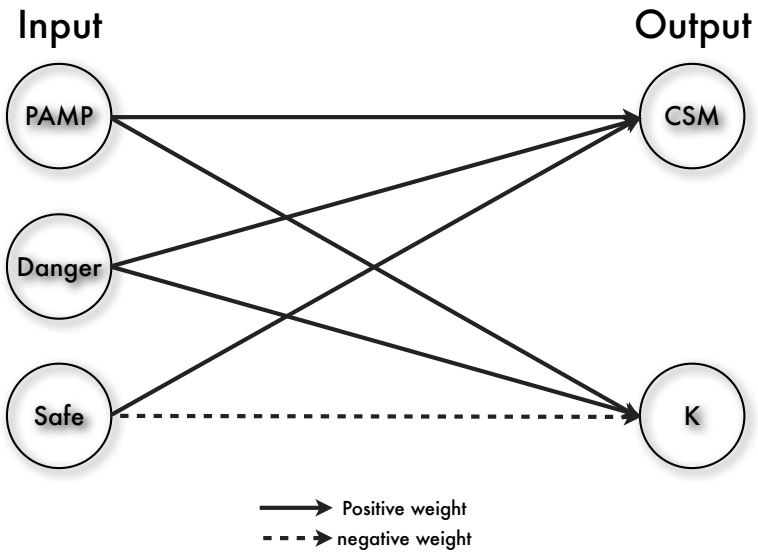

Figure 2: An illustration of the signal transformation process of the DCA.

In order to achieve its detection ability, the DCA initialises a population of artificial DCs operating in parallel as detectors. Each DC is given a distinct limit of its lifespan, which creates a dynamic time window effect in the population [26]. This leads to the same signal and antigen data streams being processed by every DC, during different time periods across the analysed time series. A temporal correlation between signals and antigens is also performed by each DC internally, to capture the causal relationship within the data. As suggested in [12], to perform correct correlation, the signals are supposed to appear after the antigens, and the delay should be shorter than the time window created by each DC.

During detection, each individual DC updates its antigen profile by storing the sampled antigens internally. In the meantime, the output signals produced by the signal transformation are accumulated, to update the DC's lifespan and signal profile. The DC's lifespan is subtracted by the cumulative *CSM*, which gives the difference between the amount of information initially allowed for a DC and that has been processed by the DC so far. Such difference reflects to if the DC has processed sufficient information and is ready to make decisions. On the other hand, its signal profile is added by the cumulative K, to aggregate the polarisation towards anomaly or nor-

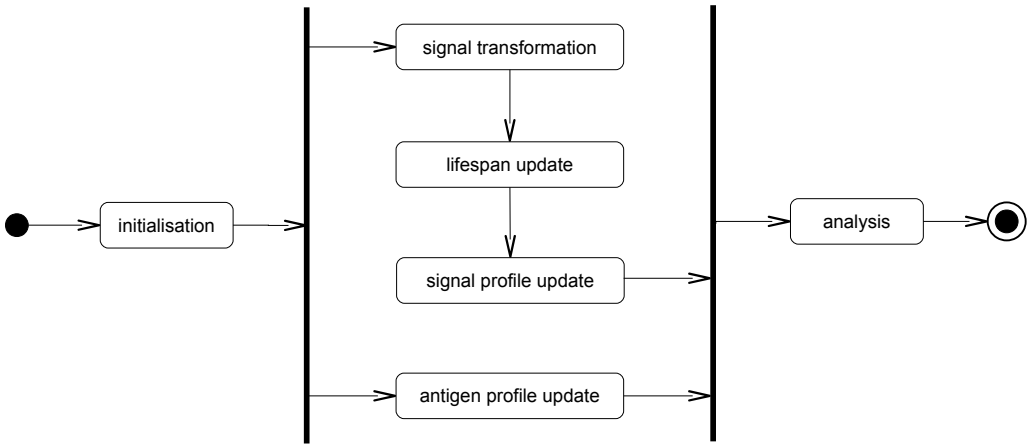

Figure 3: An illustration of different steps of the DCA, where the initialisation and analysis steps are performed at the population level and the rest of the steps (bounded within the two vertical lines) are performed at the individual DC level.

mality indicated by its tendency toward $-\infty$ or $+\infty$. As soon as the DC's lifespan reaches zero, it stops performing signal transformation and temporal correlation. The association between the cumulative K and sampled antigens within the DC, termed 'processed information', is then accumulated by the analysis phase to produce the final detection results. Once a matured DC has presented the processed information, it is reset to its default form. Here, the population size is generally kept constant, but can be user specified. The entire process of different steps of the DCA is illustrated in Figure 3.

3. Formalisation of the DCA

In this section, we formally define data structures and procedural operations of the DCA at the population level. Unlike specifications of a single DC at the behavioural level in [15], here we focus on specifying the entire DC population using quantitative measures at the functional level. Instead of using more advanced and possibly more complex interval temporal logic [23], set theory and mathematical functions e.g. addition, multiplication and recursion are used for clarity. This aims to present the algorithm in a comprehensive way, which can be easily accessed by readers who may not be familiar with formal logic.

3.1. Data Structures

Define $\mathsf{Signal} \subseteq \mathbb{R}^m$ and $\mathsf{Antigen} \subseteq \mathbb{N}$ as the two types of input data. Within a discrete time space $\mathsf{Time} = \{1, 2, \ldots, t, \ldots\}$, the input data can be defined as a function $S : \mathsf{Time} \rightarrow \mathsf{Signal} \cup \mathsf{Antigen}$, and $S(t)$ is a data instance at a time point $t \in \mathsf{Time}$. Elements from Signal are input signal instances of the algorithm, and are represented as m-dimensional real-valued vectors. These are usually normalised into a non-negative range, e.g. $[0, 1]$, as the input to the DCA. In many applications, $m = 3$ is the standard case, corresponding to the three input signal categories of the DCA as described in Section 2. Elements from $\mathsf{Antigen}$ are categorical identifiers of certain objects to be classified, and are often represented as natural numbers starting from one, where the order is ignored.

Define the weight matrix of signal transformation as

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ w_{21} & \cdots & w_{2m} \end{bmatrix}$$

where each entry $w_{ij} \in \mathbb{R}$. The weight matrix $\mathbf{W}$ is used to transform the m-dimensional input signals to two categories of output signals, namely 'CSM' and 'K'. It is usually predefined by users and kept constant during runtime. Entries in the weight matrix are based on empirical results from the underlying immunology of natural DCs [10].

Let $\mathsf{Population}$ be an index set of DCs and $N = |\mathsf{Population}|$ be the population size ($N = 100$ is a popular choice). The index of a DC is $i \in \mathsf{Population}$. The function of assigning the initial lifespan to a DC is defined as $I : \mathsf{Population} \rightarrow \mathbb{R}$, where $I(i) \neq I(j)$ $(i \neq j \in \mathsf{Population})$. The function of initialising the antigen profile of the DC is defined as $M : \mathsf{Population} \rightarrow (a_{i1}, a_{i2}, ..., a_{ik}, ...)$, where $(a_{i1}, a_{i2}, ..., a_{ik}, ...)$ is a sequence storing the antigen instances sampled by a DC and $a_{ik} \in \mathsf{Antigen}$. The initial signal profile of a DC is usually set to zero.

The output of each DC is stored as a pair $(a_{ik}, r_i) \in \mathsf{Antigen} \times \mathbb{R}$ in a list, where r_i is the signal profile of a DC when it reaches a termination condition. We also define π_1 and π_2 as projection functions to obtain the first and second elements of a pair respectively.

3.2. Procedural Operations

To access the data structures of the DCA, a series of one-step procedural operations are executed. Formally defining these operations is essential for

the algorithm's runtime analysis. At the beginning ($t = 1$), the algorithm initialises all the DCs indexed by Population, through assigning the initial values of lifespans and signal profiles, named '**DC initialisation**'. The value of $I(i)$ depends on the distribution function used to generate the initial lifespans of DCs. Both uniform distribution and Gaussian distribution can be applied to generate $I(i)$. The antigen profile of each DC is set as Null or empty, while the signal profile is set as zero.

Definition 1 (**signal transformation**). *The signal transformation function* $O : \mathsf{Time} \to \mathbb{R} \times \mathbb{R}$ *is defined as*

$$O(t) = \begin{cases} \mathbf{W}^{\mathrm{T}} S(t), & \text{if } S(t) \in \mathsf{Signal}; \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

This operation is executed whenever $S(t) \in \mathsf{Signal}$ holds, and it performs the multiplication between a transposed $2 \times m$ matrix and an m-dimensional vector to produce a two dimensional vector of output signals, 'CSM' and 'K'. These are related to when and how to make decisions respectively. In the case that $S(t) \in \mathsf{Antigen}$, the function returns a zero vector.

Definition 2 (**lifespan update**). *The lifespan update function* $F : \mathsf{Time} \times \mathsf{Population} \to \mathbb{R}$ *is defined as*

$$F(t, i) = \begin{cases} I(i), & \text{if } t = 1; \\ I(i) - \pi_1(O(t)), & \text{if } F(t-1, i) \leq 0; \\ F(t-1, i) - \pi_1(O(t)), & \text{otherwise.} \end{cases}$$

When $t = 1$, the initial value of F is $I(i)$, which is the initial lifespan of the DC with an index i. It is repeatedly subtracted by CSM signal until the termination condition, $F(t-1, i) \leq 0$, is reached. The function is then reset to '$I(i) - \pi_1(O(t))$' (not $I(i)$), due to the function $O(t)$ being executed at a regular basis, e.g. at every single time point t.

Definition 3 (**signal profile update**). *The signal profile update function* $G : \mathsf{Time} \times \mathsf{Population} \to \mathbb{R}$ *is defined as*

$$G(t, i) = \begin{cases} 0, & \text{if } t = 1; \\ \pi_2(O(t)), & \text{if } F(t-1, i) \leq 0; \\ G(t-1, i) + \pi_2(O(t)), & \text{otherwise.} \end{cases}$$

When $t = 1$, the value of G is zero, which is the initial signal profile of the DC with an index i. It is repeatedly increased by the K signal until the

termination condition is reached. The function is then reset to '$\pi_2(O(t))$' (not 0), due to the function $O(t)$ being executed at a regular basis, e.g. at every single time point $t \in \mathsf{Time}$.

Definition 4 (**antigen profile update**). *The antigen profile update function* $H : \mathsf{Time} \times \mathsf{Population} \to (a_{i1}, a_{i2}, \ldots, a_{ik}, \ldots)$ *is defined as*

$$H(t,i) = \begin{cases} \emptyset, & t = 1; \\ (H(t-1,i), S(t)), & \text{if } S(t) \in \mathsf{Antigen} \text{ and } t > 1; \\ H(t-1,i) & \text{if } S(t) \in \mathsf{Signal} \text{ and } t > 1, \end{cases}$$

where H is initially empty. As a new antigen instance arrives, it is sampled by the DC with an index i and its antigen profile is updated until the termination condition is reached. This function merely appends a list to another, which can be done in constant time regardless of the length of the lists, and thus considered as one-step operation as well. It is performed individually by each DC and the index of the DC selected to sample an incoming $S(t) \in \mathsf{Antigen}$ is defined as $i \equiv \theta \mod N$ (i is congruent with θ modulo N), where θ is the number of antigen instances up to time t. This is termed the 'sequential sampling' rule.

Definition 5 (**output record**). *Let* $r_i = G(t,i)$ s.t. $F(t-1,i) \leq 0$ *be the signal profile of a DC, and* $L : \mathbb{N} \to \mathsf{Antigen} \times \mathbb{R}$ *denote the function that maps an index* $j \in \mathbb{N}$ *to an element of the output list. The output record function is defined as*

$$L(j) = (a_{ik}, r_i) \quad \forall k$$

where $L(j)$ is the jth element of the list. The antigen profile often consists of multiple values while the signal profile only contains one single value in the DC with an index i. This function essentially enumerates all the possible pairs and appends them to the output list, where each of them is assigned an index j. The list is then used to produce the final detection results in the analysis phase of the DCA.

Definition 6 (**antigen counter**). *The antigen counter function* $C : \mathbb{N} \times \mathsf{Antigen} \to \{0,1\}$ *is defined as*

$$C(j,\alpha) = \begin{cases} 1, & \text{if } \pi_1(L(j)) = \alpha; \\ 0, & \text{otherwise.} \end{cases}$$

Definition 7 (**signal profile abstraction**). *The signal profile abstraction function* $R : \mathbb{N} \times \mathsf{Antigen} \to \mathbb{R}$ *is defined as*

$$R(j, \alpha) = \begin{cases} \pi_2(L(j)), & \text{if } \pi_1(L(j)) = \alpha; \\ 0, & \text{otherwise.} \end{cases}$$

In the two functions above, $\alpha \in \mathsf{Antigen}$ is an antigen type. The function C is used to count the number of instances of antigen type α, and the function R is used to calculate the sum of all K values associated with antigen type α. These two operations are performed for every antigen type and involve scanning the sequence of $L(j)$ in its entirety.

Definition 8 (**anomaly metric calculation**). *Given the number of input instances is equal to n, the anomaly metric calculation function is defined as.*

$$K(\alpha) = \frac{\gamma}{\beta} \text{ with } \beta = \sum_{j=1}^{n} C(j, \alpha) \text{ and } \gamma = \sum_{j=1}^{n} R(j, \alpha)$$

As $\mathsf{Antigen} \neq \emptyset$ and $\alpha \in \mathsf{Antigen}$, the minimum number of antigen instances is equal to one, so is the minimum number of antigen types. Therefore, we have $\beta \geq 1$. A threshold ε can be applied for further classification. The value of the threshold depends on the underlying characteristics of the dataset used. An antigen type α is classified as anomalous if $K(\alpha) > \varepsilon$, and normal otherwise.

4. Analysis of Runtime Complexity

4.1. The Standard DCA

By combining the procedural operations of the DCA with **for**, **while** loops or **if** statements the algorithm can be presented as in Algorithm 1. Previous applications of the DCA have shown that the runtime of the algorithm is relatively short and the consumption of computation power is also low [12]. However, theoretical analysis of the runtime complexity of the DCA, given a set of input data, has not yet been performed. Runtime analysis involves calculating the number of primitive operations or steps executed by an algorithm [5]. The analysis is often based on asymptotic theory, and its aim is to theoretically show the runtime complexity of an algorithm as a function of increasing input size n.

As mentioned previously, applications of the DCA are referred to the area of anomaly detection. In AIS, one popular anomaly detection algorithm is known as the negative selection algorithm, which was shown to have an exponential runtime complexity [30]. An attempt of reducing the worst-case runtime complexity from exponential to polynomial was reported in [9], however this reduction is only applicable when the input feature space is bit strings instead of real numbers. Other popular anomaly detection algorithms are more or less derived from techniques in machine learning [4], e.g. K-Nearest Neighbour (KNN) with a runtime complexity of $\mathcal{O}(nd)$ [8], decision trees algorithms with an exponential runtime complexity [8], and support vector machines (SVM) with a runtime complexity of $\mathcal{O}(n^2d)$ [3], where n is the number of input instances and d is the dimensionality. As a result, the subsequent runtime analysis of the DCA reveals if the algorithm is competitive against other state-of-the-art anomaly detection algorithms.

Let a be the number of antigen instances within the input data, $b = |\mathsf{Antigen}|$ be the number of antigen types and N be the size of the DC population. According to previous applications [1, 25, 11, 22, 17], N is usually user defined and independent of the increase of data size n. However, we often assume that $1 \leq N \leq n$. In order to make the following analyses more general, the population size N is considered a parameter of the algorithm. As the type of input data instances is either Antigen or Signal, if the number of antigen instances is equal to a, the number of signal instance is $n - a$. For the ease of analysis, the algorithm is divided into three phases as follows:

1. `Initialisation phase` - Line 1 to Line 3;
2. `Detection phase` - Line 4 to Line 19;
3. `Analysis phase` - Line 20 to Line 26.

The calculation of runtime is performed phase by phase. Let $T_1(n)$, $T_2(n)$ and $T_3(n)$ be the runtime of each phase respectively, and $T(n) = T_1(n) + T_2(n) + T_3(n)$ is the overall runtime of the algorithm. Details of all the primitive operations of the algorithm are listed in Table 1, including the **line number** and the **description** of each operation as well as the **number of times** an operation is executed, corresponding to Algorithm 1.

The `initialisation phase` is only executed once for the entire DC population at the commencement of the algorithm. Its runtime is independent of the number of input instances n, but is determined by the population size N. Therefore, the runtime of the `initialisation phase` is calculated as

follows.

$$T_1(n) = N + N = \mathcal{O}(N)$$

The runtime of the `detection phase` depends on the data size n, the number of antigen instances a, the number of signal instances $n - a$ and the size of the DC population N. Thus the runtime of the `detection phase` is calculated as follows.

$$\begin{aligned} & T_2(n) = n + 3a + 2(n-a) + 5(n-a)N = 3n + 5N(n-a) + a \\ \Rightarrow \quad & \{a \leq n\} \\ & T_2(n) = \mathcal{O}(n) + \mathcal{O}(N(n-a)) + \mathcal{O}(a) = \mathcal{O}(nN) \end{aligned}$$

The runtime of the `analysis phase` is dependent on the size of the output list that is equal to the number of antigen instances a and the number of antigen types b. The value of b is determined by the number of states or entities to classify within a problem domain. Here we merely focus on the worst-case scenario, which occurs if $b = a$, and the number of antigen types is equal to the number of antigen instances. Therefore, we have $1 \leq b \leq a \leq n$. The runtime of the `analysis phase` is thus calculated as follows.

$$T_3(n) = a + ab + 3ab = \mathcal{O}(n^2)$$

Theorem 1. *The runtime complexity of the standard DCA is bounded by $\mathcal{O}(n^2)$, with respect to the data size n.*

Proof.

$$\begin{aligned} & T(n) = T_1(n) + T_2(n) + T_3(n) \\ \Rightarrow \quad & \{T_1(n) = \mathcal{O}(N),\ T_2(n) = \mathcal{O}(nN),\ \text{and } T_3(n) = \mathcal{O}(n^2)\} \\ & T(n) = \mathcal{O}(N) + \mathcal{O}(nN) + \mathcal{O}(n^2) \\ \Rightarrow \quad & \{1 \leq N \leq n\} \\ & T(n) = \mathcal{O}(n^2) \end{aligned}$$

Bounds provided by $\mathcal{O}$-notation are asymptotically tight. □

As suggested by Theorem 1, the DCA has a worst case runtime complexity of $\mathcal{O}(n^2)$, which is quadratic. As a result, the DCA is indeed competitive in terms of processing large-sized datasets while keeping the runtime complexity under control, when compared to state-of-the-art anomaly detection

algorithms. According to previous applications [1, 11, 25, 22, 17, 14], we often have $N \ll n$. Such a premise makes the runtime complexity of algorithm's initialisation and detection phases overall linear, while the analysis phase stays quadratic. This leads to the following work of modifying the analysis phase of the algorithm via an introduction of segmentation.

4.2. The DCA with Segmentation

Segmentation is introduced to adapt the algorithm to online analysis [16]. Instead of analysing the processed information in a single operation at the termination of the detection phase, the output list is partitioned into smaller segments and the analysis is performed within each segment. We postulate that segmentation could potentially generate finer grained results, as well as performing analysis in parallel with the detection process. Here, we focus on the antigen based segmentation approach, as it is more favourable in actual applications [16]. One may think that the system with segmentation produces the final detection results much faster, as the analysis is performed during detection on a much smaller chunk of processed information. Based on the analysis of the standard DCA, it is possible to theoretically analyse the effect of segmentation on the algorithm's runtime complexity. Let z be a predefined segment size and $1 \leq z \leq n$. A segment is generated once the size of the output list reaches z, and an analysis on the current batch of processed information in the output list is performed.

As a post-processing mechanism, segmentation only affects the `analysis phase` of the algorithm, but not the `initialisation phase` or `detection phase`. The search space of the analysis of a segment is determined by the value of z. The number of segments created is equal to $\lceil n/z \rceil$, and they are indexed by $\{1, 2, \ldots, k, \ldots, \lceil n/z \rceil\}$. Let $a_k \leq z$ and $b_k \leq z$ denote the number antigen instances and the number of antigen types in the kth segment respectively. As a result, the runtime complexity of each segment at the analysis phase is $T_3^k(n) = a_k b_k \leq z^2 = \mathcal{O}(z^2)$.

Theorem 2. *The runtime complexity of the DCA with segmentation is bounded by* $\mathcal{O}(\max(nN, nz))$*, with respect to the data size* n*, the DC population size* N*, and the segment size* z*.*

Proof.

$$
\begin{aligned}
\Rightarrow \quad & \{T_1(n) = \mathcal{O}(N) \text{ and } T_2(n) = \mathcal{O}(nN)\} \\
& T(n) = \mathcal{O}(N) + \mathcal{O}(nN) + \sum_{k=1}^{\lceil n/z \rceil} a_k b_k \leq \mathcal{O}(N) + \mathcal{O}(nN) + \lceil n/z \rceil \mathcal{O}(z^2) \\
\Rightarrow \quad & \{1 \leq N \leq n \text{ and } 1 \leq z \leq n\} \\
& T(n) = \mathcal{O}(N) + \mathcal{O}(nN) + \mathcal{O}(nz) = \mathcal{O}(\max(nN, nz))
\end{aligned}
$$

□

As shown in Theorem 2, the introduction of segmentation changes the overall runtime complexity of the algorithm to $\mathcal{O}(\max(nN, nz))$. Depending on the values of N and z, the runtime complexity can be either quadratic ($N = n \vee z = n$) or linear ($N \ll n \wedge z \ll n$). This is very attractive for online detection tasks, as it provides a means of online analysis that continuously and periodically produces results during detection. Additionally, the DCA with segmentation produces significantly different and better results than the standard version [16]. Therefore, segmentation is an important and necessary addition to the DCA from a practical point of view. Thus far only static segmentation with a fixed segment size has been applied to the DCA. The effect of variable segment sizes on the detection performance still requires further investigation.

5. Formulation of Runtime Properties

Two runtime variables of the DCA are assessed, as they can be used as quantitative indicators of the changes to the algorithm's runtime behaviour. They are the number of matured DCs (those which reach the termination condition and are reset) and the number of processed antigens respectively. The number of matured DCs indicates that the amount of processed information is related to signal instances. Conversely, the number of processed antigens implies that the amount of processed information is related to antigen instances. In this section, the formulation of the above properties is given, to build up the mathematical foundation of the algorithm. This is obtained with respect to a time interval $[t_b, t_e] := \{t_b, t_b + 1, t_b + 2, \ldots, t_e\} \subseteq \mathsf{Time}$.

5.1. Number of Matured DCs

The number of matured DCs within a time interval is related to the reset frequency of the DC population, which indicates the work-load of the DC population. This can be used to determine whether the current setup of the current system should be altered. If the frequency of DC resetting is too high, most of the DCs become matured and get reset before they acquire a sufficient amount of information. As a result, the range of lifespans of the DC population should be extended, allowing more information to be obtained. In conduction with extending the range of lifespans of the DC population, it is necessary to also increase the size of the DC population, so that the lifespans do not become sparse.

This becomes crucial if the system is deployed online, as an online system is often required to perform continuous detection and adapt to the changes of real-time situations. The number of matured DCs in the DC population depends on the distribution function used for the generation of DC lifespans, in addition to the input data within the time interval of interest. To make the analysis manageable, two types of distributions for generating the initial DC lifespans are considered, namely uniform distribution [2] and Gaussian distribution [2]. The calculations will be done through using the mean value of lifespans of the DC population and the mean value of CSM signals corresponding to all the input signal instances. They focus on the average number of matured DCs within a given time interval rather than the particular number per iteration. However, as the time interval is reduced, e.g. to the duration of one iteration, the two numbers could become approximate to each other.

Proposition 1 (uniform distribution)**.** *If the lifespans of the DC population are generated from an arithmetic series $x_i = x_1 + (i-1)d$, where x_i is the* nth *element, x_1 is the first element and d is the interval between two successive elements, the number of matured DCs in the DC population δ can be calculated as follows.*

$$\delta = \left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(t_e - t_b)(x_1 + \frac{N-1}{2}d)} \right\rfloor$$

By default, the ascending order of lifespans of the DC population corresponds to the order of its indices. As a result, if the size of the DC population

is equal to N, the lifespan of the last DC with an index $i = N$ is given as $x_N = x_1 + (N-1)d$. As demonstrated in Section 3, the termination condition where a DC matures as soon as its lifespan reaches zero through subtracting the CSM signals.

Proof.

$$\Rightarrow \quad \{\varphi = \frac{\sum_{t=t_b}^{t_e} \pi_1(O(t))}{t_e - t_b} \text{ and } \mu_1 = \frac{x_1 + x_N}{2} = x_1 + \frac{N-1}{2}d\}$$
$$\delta = \left\lfloor \frac{N\varphi}{\mu_1} \right\rfloor = \left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(t_e - t_b)(x_1 + \frac{N-1}{2}d)} \right\rfloor$$

Where φ is the mean value of the *CSM* signals within the interval $[t_b, t_e]$ and μ_1 is the mean lifespan of the DC population. □

Uniform distribution is used in the dDCA [12] to generate the initial lifespans of the DC population. This produces a set of values that are uniformly distributed within a certain range. According to Proposition 1, if the parameters (first element x_1 and the interval d) of the arithmetic series are given, the number of matured DCs within the time interval $[t_b, t_e]$ can be calculated accordingly.

Proposition 2 (Gaussian distribution)**.** *If the lifespans of the DC population are generated from a Gaussian distribution $x \sim \mathcal{N}(\mu, \sigma^2)$, then the following formula holds.*

$$\Pr\left(\left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(\mu - \frac{2\sigma}{\sqrt{N}})(t_e - t_b)} \right\rfloor \leq \delta \leq \left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(\mu + \frac{2\sigma}{\sqrt{N}})(t_e - t_b)} \right\rfloor \right) = 0.95$$

Proof.

$$
\begin{aligned}
\Rightarrow \quad & \{\varphi = \frac{\sum_{t=t_b}^{t_e} \pi_1(O(t))}{t_e - t_b} \text{ and } \mu_2 \sim \mathcal{N}(\mu, \frac{\sigma^2}{N})\} \\
& \Pr\left(\mu - 2\frac{\sigma}{\sqrt{N}} \leq \mu_2 \leq \mu + 2\frac{\sigma}{\sqrt{N}}\right) = 0.95 \\
\Rightarrow \quad & \{\delta = \left\lfloor \frac{N\varphi}{\mu_2} \right\rfloor = \left\lfloor \frac{N}{\mu_2(t_e - t_b)} \sum_{t=t_b}^{t_e} \pi_1(O(t)) \right\rfloor\} \\
& \left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(\mu - \frac{2\sigma}{\sqrt{N}})(t_e - t_b)} \right\rfloor \leq \delta \leq \left\lfloor \frac{N \sum_{t=t_b}^{t_e} \pi_1(O(t))}{(\mu + \frac{2\sigma}{\sqrt{N}})(t_e - t_b)} \right\rfloor
\end{aligned}
$$

$\Pr(\cdot)$ is the probability operator. If the sample size is N, the sample mean μ_2 is bounded by a Gaussian distribution $x \sim \mathcal{N}(\mu, \frac{\sigma^2}{N})$ [2]. The lower and upper bounds of the sample mean can be used to induce the bounds of the number of matured DCs. □

In practice, Gaussian distribution has not been used for generating the lifespans of the DC population, but it has been of great interest [27] and would be a priority of future investigation. According to Proposition 2, if we know the mean (μ) and variance (σ^2) of the Gaussian distribution from which the lifespans of the DC population are generated, the size of DC population N, and the input data instances within the time interval $[t_b, t_e]$, we can show that there is a 0.95 chance the number of matured DCs is bounded by the lower and upper bounds. This could provide sufficient information for adjusting the system according to real-time scenarios.

5.2. Number of Processed Antigens

As demonstrated in [16], segmentation is effective for maintaining or even improving detection accuracy on large-sized datasets. This may be due to the fact that the number of processed antigens could determine whether an analysis of the current batch of processed information is required. Different from input antigen instances, processed antigens are those, presented by matured DCs. Investigation of the relationship between the number of processed antigens and the input data becomes essential for understanding the DCA, as well as for the development of integrating segmentation with the algorithm. Additionally, a priori knowledge of the number of processed antigens, based on the input data, may facilitate choosing an appropriate segment size. Here,

we focus on formulating the relationship between the number of processed antigens and the input data, in particular, the number of input antigens. Let $\theta \in \mathbb{N}$ at $t \in \mathsf{Time}$ be the number of input antigens that are fed into the system, and δ be the mean lifespan of the DC population. Similar to Proposition 1 or Proposition 2, the calculations focus on the average number of processed antigens within a given time interval rather than the particular duration per iteration. As the time interval decreases, e.g. to the duration of one iteration, the two numbers could also be approximate to each other.

The method of calculating the number of processed antigens within a given time interval $[t_b, t_e]$ should be introduced first. It is similar to placing balls into a number of bins that are ordered based on their indexes in a sequential manner. Placing starts from the first bin then the second bin and so forth. If we reach the last bin, the process starts over again. In the end, a number of bins, starting from the first one, are taken and the number of balls contained is counted. The balls are equivalent to input antigens, the bins are equivalent to DCs, and the action of counting the number of balls is equivalent to the action of counting the number of processed antigens. Proposition 3 formulates the relationship between the number of processed antigens and the input data in two cases.

Proposition 3 (**number of processed antigens**). *Let ν be the number of processed antigens within a given interval $[t_b, t_e]$, $c \equiv \delta \mod N$ and $d \equiv \theta \mod N$, the following formula of ν holds.*

$$\nu = \begin{cases} \left(\delta - N \left\lfloor \frac{\delta}{N} \right\rfloor\right)\left(1 + \left\lfloor \frac{\theta}{N} \right\rfloor\right), & \text{if } c < d; \\ \left(\delta - N - N \left\lfloor \frac{\delta}{N} \right\rfloor\right) \left\lfloor \frac{\theta}{N} \right\rfloor + \theta, & \text{otherwise.} \end{cases}$$

Proof.

$$
\begin{aligned}
& \{\text{transform modulus to floor functions}\} \\
\Rightarrow \quad & c = \delta - N \left\lfloor \frac{\delta}{N} \right\rfloor \text{ and } d = \theta - N \left\lfloor \frac{\theta}{N} \right\rfloor \\
\Rightarrow \quad & \{\text{sequential sampling}\} \\
& \text{Case } 1 : c < d \\
& \nu = c \left\lfloor \frac{\theta}{N} \right\rfloor + c = \left(\delta - N \left\lfloor \frac{\delta}{N} \right\rfloor \right) \left(1 + \left\lfloor \frac{\theta}{N} \right\rfloor \right) \\
& \text{Case } 2 : c \geq d \\
& \nu = c \left\lfloor \frac{\theta}{N} \right\rfloor + d = \left(\delta - N - N \left\lfloor \frac{\delta}{N} \right\rfloor \right) \left\lfloor \frac{\theta}{N} \right\rfloor + \theta
\end{aligned}
$$

The number of antigens sampled by each DC is determined by $\theta \mod N$, but as only matured DCs present processed antigens, the number of processed antigens is determined by δ and the maximum of c and d. □

The formulas for the number of processed antigens have two cases, depending on the relationship between $c \equiv \delta \mod N$ and $d \equiv \theta \mod N$. These formulas can relate the runtime variables of the algorithm to the input data, without actually running the algorithm. This provides theoretical insights into tuning the algorithm for a given problem.

6. Conclusions and Future Work

In this paper, we provide formal definitions of the data structures and procedural operations of the deterministic version of the DCA, name the dDCA. It aims to clearly present the algorithm, to prevent future misunderstanding and ambiguity that could result in inappropriate applications and implementations. Based on the formal definitions, a runtime analysis of the standard DCA is performed. The DCA achieves the the worst-case runtime complexity bounded by $\mathcal{O}(n^2)$, which is quadratic. The analysis of the system with segmentation is also performed. We have shown that the introduction of segmentation does change the algorithm's runtime complexity and in certain cases it approximates to linear. In addition, it provides a means of performing continuous and periodic analysis for the DCA. This makes the algorithm very attractive for online detection tasks.

Moreover, two runtime variables are formulated, the number of matured DCs and the number of processed antigens. This shows how the algorithm behaves within a given time interval based on the input data without actually running the algorithm. As a result, the formulas of two runtime variables can be used as the indicators of adjusting the setup of the system according to real-time situations during detection. This an important step for understanding some of the potentially beneficial properties of the algorithm from a theoretical perspective, which could facilitate further investigations on the usefulness of these properties with respect to anomaly detection problems.

This work gives application independent insights to the algorithm, which can be used as guidelines for future development. One of the goals of future development of the DCA is to turn it into an automated and adaptive online detection system, and such a system has certain requirements to fulfil. Firstly, the system has to be computationally efficient. The analysis of the runtime complexity of the DCA shows even in worst case scenarios its runtime complexity is competitive against other popular anomaly detection algorithms. Secondly, the system should be able to adapt to real-time scenarios encountered during detection. This requires the insights of how the algorithm behaves during runtime, which can be assessed from the two runtime variables. As a result, new components can be developed and integrated within the algorithm to adjust the system based on the assessment of these two runtime variables.

In terms of future work, the specifications can be further simplified and the algorithm can be presented using functional programming approach [6], to reveal more algorithmic details. In addition, synthetic datasets generated from various probability density functions can be used to test the formulas defined in this paper. We can also investigate other properties of the algorithm, for example, the moving window effect created by each DC and the relationship between the size of DC population and the detection performance. Different methods of generating the initial lifespans of the DC population should also be investigated, in addition to the relationship between the weight matrix and the detection performance.

References

[1] Y. Al-Hammadi, U. Aickelin, and J. Greensmith. DCA for Bot Detection. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI)*, pages 1807–1816, 2008.

[2] A. C. Atkinson, M. Riani, and A. Cerioli. *Exploring Multivariate Data with the Forward Search*. Springer Series in Statistics. Springer, 2004.

[3] C. J. C. Buerges. A Tutorial on Support Vector Mahinces for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):Article 15, 2009.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition (Hardcover)*. The MIT Electrical Engineering and Computer Sicence Series. The MIT Press, 2009.

[6] G. Cousineau, M. Mauny, and K. Callaway. *The Functional Approach to Programming*. Cambridge University Press, 1998.

[7] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligent Approach*. Springer-Verlag, 2002.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Blackwell, 2nd edition, 2000.

[9] M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. *Theoretical Computer Sicence*, 412(6):534–542, 2011.

[10] J. Greensmith. *The Dendritic Cell Algorithm*. PhD thesis, School of Computer Science, University of Nottingham, 2007.

[11] J. Greensmith and U. Aickelin. DCA for SYN Scan Detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 49–56, 2007.

[12] J. Greensmith and U. Aickelin. The Deterministic Dendritic Cell Algorithm. In *Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS)*, pages 291–303, 2008.

[13] J. Greensmith, J. Feyereisl, and U. Aickelin. The DCA: SOMe Comparison A comparative study between two biologically-inspired algorithms. *Evolutionary Intelligence*, 1(2):85–112, 2008.

[14] F. Gu, J. Greensmith, and U. Aickelin. Further Exploration of the Dendritic Cell Algorithm: Antigen Multiplier and Time Windows. In *Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS)*, pages 142–153, 2008.

[15] F. Gu, J. Greensmith, and U. Aickelin. Exploration of the Dendritic Cell Algorithm with the Duration Calculus. In *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS)*, pages 54–66, 2009.

[16] F. Gu, J. Greensmith, and U. Aickelin. Integrating Real-Time Analysis With The Dendritic Cell Algorithm Through Segmentation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1203–1210, 2009.

[17] F. Gu, J. Greensmith, R. Oates, and U. Aickelin. PCA 4 DCA: the application of Principal Component Analysis to the Dendritic Cell Algorithm. In *Proceedings of the 9th Annual Workshop on Computational Intelligence (UKCI)*, 2009.

[18] E. Hart and J. Timmis. Application Areas of AIS: The Past, Present and the Future. *Journal of Applied Soft Computing*, 8(1):191–201, 2008.

[19] T. Janson and C. Zarges. Analyzing different variants of immune inspired somatic contiguous hypermutations. *Theoretical Computer Sicence*, 412(6):517–533, 2011.

[20] A. Jean-Raymond. *The B-Book*. Cambridge University Press, 1996.

[21] M. B. Lutz and G. Schuler. Immature, semi-mature and fully mature dendritic cells: which signals induce tolerance or immunity? *TRENDS in Immunology*, 23(9):445–449, 2002.

[22] M. Mokhtar, R. Bi, J. Timmis, and A. M. Tyrrell. A Modified Dendritic Cell Algorithm for On-line Error Detection in Robotic Systems. In *Proceedings of the 11th IEEE Congress on Evolutionary Computation (CEC)*, pages 2055–2062, 2009.

[23] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *Computer*, 18(2):10–19, 1985.

[24] R. Oates. *The Suitability of the Dendritic Cell Algorithm for Robotic Security Applications*. PhD thesis, School of Computer Science, University of Nottingham, 2010.

[25] R. Oates, J. Greensmith, U. Aickelin, J. Garibaldi, and G. Kendall. The Application of a Dendritic Cell Algorithm to a Robotic Classifier. In *Proceedings of the 6th International Conference on Artificial Immune (ICARIS)*, pages 204–215, 2007.

[26] R. Oates, G. Kendall, and J. Garibaldi. Frequency Analysis for Dendritic Cell Population Tuning: Decimating the Dendritic Cell. *Evolutionary Intelligence*, 1(2):145–157, 2008.

[27] R. Oates, G. Kendall, and J. Garibaldi. Classifying in the presence of uncertainty: a DCA persepctive. In *Proceedings of the 9th International Conference on Artificial Immune Systems (ICARIS)*, pages 75–87, 2010.

[28] T. Stibor, R. Oates, G. Kendall, and J. Garibaldi. Geometrical insights into the dendritic cell algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1275–1282, 2009.

[29] T. Stibor, J. Timmis, and E. Claudia. A Comparative Study of Real-Valued Negative Selection to Statistical Anomaly Detection Techniques. In *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS)*, pages 262–375, 2005.

[30] J. Timmis, A. Home, T. Stibor, and E. Clark. Theoretical advances in artificial immune systems. *Theoretical Computer Sicence*, 403(1):11–32, 2008.

[31] C. Zarges. Rigorous runtime analysis of inversely fitness proportional muation rates. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, pages 112–122, 2008.

[32] C. Zarges. On the utility of the population size for inversely fitness proportional mutation rates. In *Proceedings of the 10th ACM SIGEVO Workshop on Fundations of Genetic Algorithms (FOGA)*, pages 39–46, 2009.

Algorithm 1: Pseudocode of the DCA implementation, the selection of a DC when an antigen instance is presented is performed according to the 'sequential sampling' rule.

input : input data $S(t)$
output: anomaly metric $K(\alpha)$

```
1  foreach DC do                                  /* Initialisation phase */
2  |  DC initialisation;
3  end
4  while input data do                            /* Detection phase */
5  |  if antigen then
6  |  |  select a DC i;
7  |  |  do H(t, i);
8  |  end
9  |  if signal then
10 |  |  do O(t);
11 |  |  foreach DC do
12 |  |  |  do F(t, i);
13 |  |  |  do G(t, i);
14 |  |  |  if F(t − 1, i) ≤ 0 then
15 |  |  |  |  do L(j);
16 |  |  |  end
17 |  |  end
18 |  end
19 end
20 while output list do                            /* Analysis phase */
21 |  foreach antigen type do
22 |  |  do C(j, α);
23 |  |  do R(j, α);
24 |  |  do K(α);
25 |  end
26 end
```

Line No.	Description	Times
1	**for** loop	N
2	DC initialisation	N
4	**while** loop	n
5	**if** statement	a
6	select a DC i	a
7	antigen profile update ($H(t,i)$)	a
9	**if** statement	$n-a$
10	signal transformation ($O(t)$)	$n-a$
11	**for** loop	$(n-a)\times N$
12	lifespan update ($F(t,i)$)	$(n-a)\times N$
13	signal profile update ($G(t,i)$)	$(n-a)\times N$
14	**if** statement	$(n-a)\times N$
15	output record ($L(j)$)	$(n-a)\times N$
20	**while** loop	a
21	**for** loop	$a\times b$
22	antigen counter ($C(j,\alpha)$)	$a\times b$
23	signal profile abstraction ($R(j,\alpha)$)	$a\times b$
24	anomaly metric calculation ($K(\alpha)$)	$a\times b$

Table 1: Details of primitive operations of Algorithm 1, where N is the size of DC population, n is the data size, a is the number of antigen instances, and b is the number of antigen types.

Page No.	Notation	Description
9	Signal	a set of signal instances
9	Antigen	a set of antigen instances
9	t	a time point
9	$S(t)$	a map of t to an input instance
9	$\mathbf{W}$	weight matrix of signal transformation
9	N	DC population size
9	I	an index set of DCs
9	π_1	projection function for the first element
9	π_2	projection function for the second element
10	$O(t)$	signal transformation function
10	$F(t, i)$	lifespan update function
10	$G(t, i)$	signal profile update function
11	$H(t, i)$	antigen profile update function
11	$L(j)$	output record function
11	$C(j, \alpha)$	antigen counter function
12	$R(j, \alpha)$	signal profile abstraction function
12	$K(\alpha)$	anomaly metric calculation function
12	n	size of input data
13	a	number of antigen instances
13	b	number of antigen types
15	z	segment size

Table 2: List of terms and definitions used in Section 3 and Section 4.